AADL and Model-based Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter H. Feiler
Oct 20, 2014

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **20 OCT 2014**	2. REPORT TYPE **N/A**	3. DATES COVERED
4. TITLE AND SUBTITLE **AADL and Model-based Engineering**		5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Feiler /Peter**		5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited.**		
13. SUPPLEMENTARY NOTES **The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**51**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Outline

- Challenges in Safety-critical Software-intensive systems
- An Architecture-centric Virtual Integration Strategy with SAE AADL
- Improving the Quality of Requirements
- Architecture Fault Modeling and Safety
- Incremental Life-cycle Assurance of Systems
- Summary and Conclusion

We Rely on Software for Safe Aircraft Operation

Quantas
Landing

Written by htbw
From: soyawan

Even with the autopilot off, flight control computers still ``command control surfaces to protect the aircraft from unsafe conditions such as a stall,'' the investigators said.

The unit continued to send false stall and speed warnings to the aircraft's primary computer and about 2 minutes after the initial fault ``generated very high, random and incorrect values for the aircraft's angle of attack.''

mayday call when it suddenly changed altitude during a flight from Singapore to Perth, Qantas said.

Embedded software systems introduce a new class of problems not addressed by traditional system modeling & analysis

lunge

wide

irways

causing the jet to nosedive.

was cruising at 37,000 feet (11,277 meters) when the computer fed incorrect information to the flight control system, the **Australian Transport Safety Bureau** said yesterday. The aircraft dropped 650 feet within seconds, slamming passengers and crew into the cabin ceiling, before the pilots regained control.

``This appears to be a unique event,'' the bureau said, adding that

fitted with the same air-data computer. The advisory is ``aimed at minimizing the risk in the unlikely event of a similar occurrence.''

Autopilot Off

A ``preliminary analysis'' of the Qantas plunge showed the error occurred in one of the jet's three air data inertial reference units, which caused the autopilot to disconnect, the ATSB said in a statement on its Web site.

The crew flew the aircraft manually to the end of the flight, except for a period of a few seconds, the bureau said.

Even with the autopilot off, flight control computers still ``command control surfaces to protect the aircraft from unsafe conditions such as a stall,'' the investigators said.

The unit continued to send false stall and speed warnings to the aircraft's primary computer and about 2 minutes after the initial fault ``generated very high, random and incorrect values for the aircraft's angle of attack.''

The flight control computer then commanded a ``nose-down aircraft movement, which resulted in the aircraft pitching down to a maximum of about 8.5 degrees,'' it said.

No `Similar Event'

``Airbus has advised that it is not aware of any similar event over the many years of operation of the Airbus,'' the bureau added, saying it will continue investigating.

Software Problems not just in Aircraft

May 7, 2010

Lexus GX 460 passes retest; Consumer Reports lifts "Don't Buy" label

Consumer Reports is lifting the Don't Buy: Safety Risk designation from the 2010 Lexus GX 460 SUV after recall work corrected the problem it displayed in one of our emergency handling tests. (See the original report and video: "Don't Buy: Safety Risk--2010 Lexus GX 460.")

We originally experienced the problem in a test that we use to evaluate what's called lift-off oversteer. In this test, as the vehicle is driven through a turn, the driver quickly lifts his foot off the accelerator pedal to see how the vehicle reacts. When we did this with our GX 460, its rear end slid out until the vehicle was almost sideways. Although the GX 460 has electronic stability control, which is designed to prevent a vehicle from sliding, the system wasn't intervening quickly enough to stop the slide. We consider this a safety risk because in a real-world situation this could cause a rear tire to strike a curb or slide off of the pavement, possibly causing the vehicle to roll over. Tall vehicles with a high center of gravity, such as the GX 460, heighten our concern. We are not aware, however, of any reports of injury related to this problem.

Lexus recently duplicated the problem on its own test track and developed a software upgrade for the vehicle's ESC system that would prevent the problem from happening. Dealers received the software fix last week and began notifying GX 460 owners to bring their vehicles in for repair.

We contacted the Lexus dealership from which we had anonymously bought the vehicle and made an appointment to have the recall work performed. The work took about an hour and a half.

Following that, we again put the SUV through our full series of emergency handling tests. This time, the ESC system intervened earlier and its rear did not slide out in the lift-off oversteer test. Instead, the vehicle understeered—or plowed—when it exceeded its limits of traction, which is a more common result and makes the vehicle more predictable and less likely to roll over. Overall, we did not experience any safety concerns with the corrected GX 460 in our handling tests.

Expert • Independent • Nonprofit

ConsumerReports.org

This article appeared in May 2010 Consumer Reports Magazine.

Many appliances now rely on electronic controls and operating softw... But it turned out to be a problem for the Kenmore 4027 front-loader, which scored near the bottom in our February 2010 report.

Our tests found that the rinse cycles on some models worked improperly, resulting in an unimpressive cleaning.

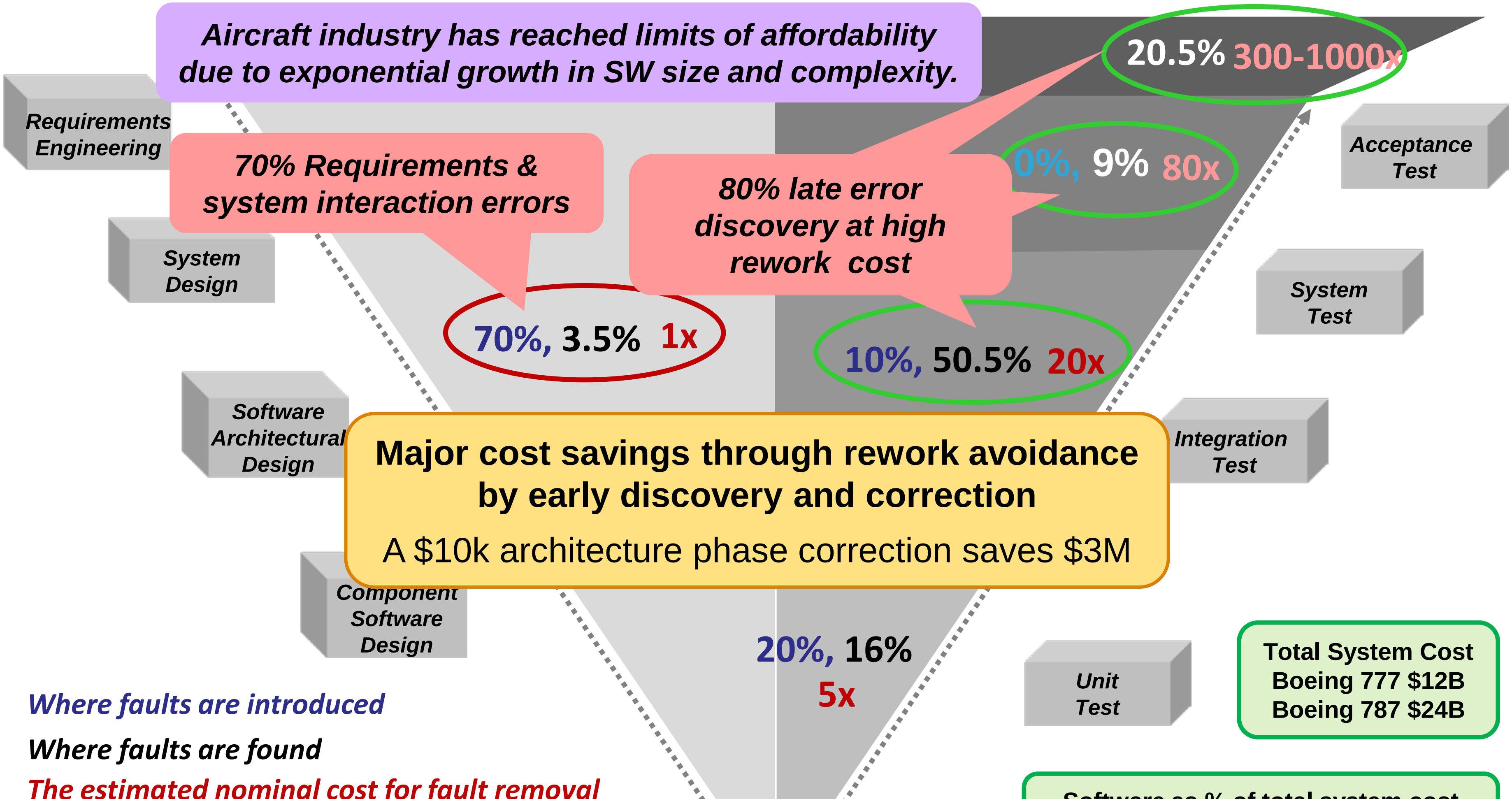
When Sears, which sells the washer, saw our February 2010 Ratings (available to subscribers), it worked with LG, which makes the washer, to figure out what was wrong. They quickly determined that a software problem was causing short or missing rinse and wash cycles, affecting wash performance. Sears and LG say they have reprogrammed the software on the models in their warehouses and on about 65 percent of the washers already sold, including the ones we had purchased.

Our retests of the reprogrammed Kenmore 4027 found that the cycles now worked properly, and the machine excelled. It now tops our Ratings (available to subscribers) of more than 50 front-loaders and we've made it a CR Best Buy.

If you own the washer, or a related model such as the Kenmore 4044 or Kenmore Elite 4051 or 4219, you should get a letter from Sears for a free service call. Or you can call 800-733-2299.

How do you upgrade washing machine software?

High Fault Leakage Drives Major Increase in Rework Cost



Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal

Sources:

NIST Planning report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, May 2002.

D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson/Addison-Wesley (2004)

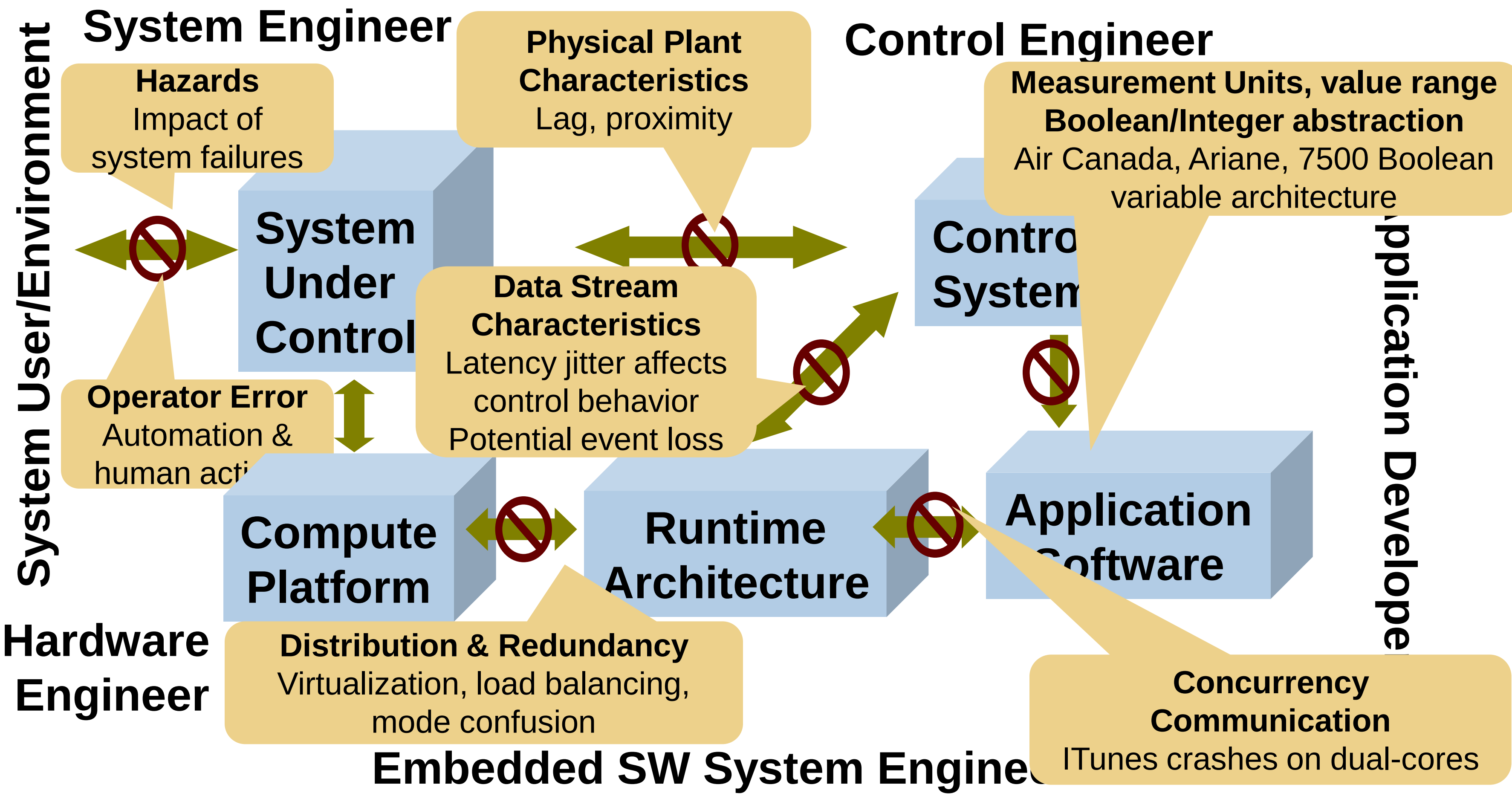
B.W. Boehm, *Software Engineering Economics*, Prentice Hall (1981)

Total System Cost
Boeing 777 $12B
Boeing 787 $24B

Software as % of total system cost
1997: 45% → 2010: 66% → 2024: 88%

Post-unit test software rework cost
50% of total system cost and growing

Mismatched Assumptions in System Interactions



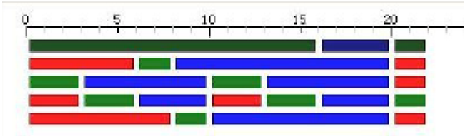
Embedded software system as major source of hazards

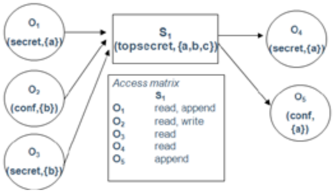
Why do system level failures still occur despite fault tolerance techniques being deployed in systems?

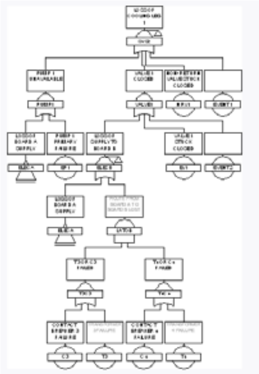
Model-based Engineering Pitfalls

The system

Inconsistency between independently developed analytical models

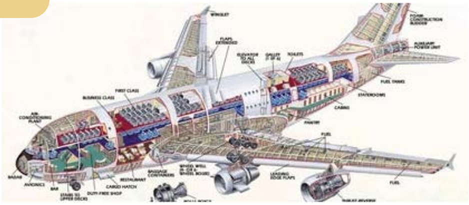






System models

Confidence that model reflects implementation



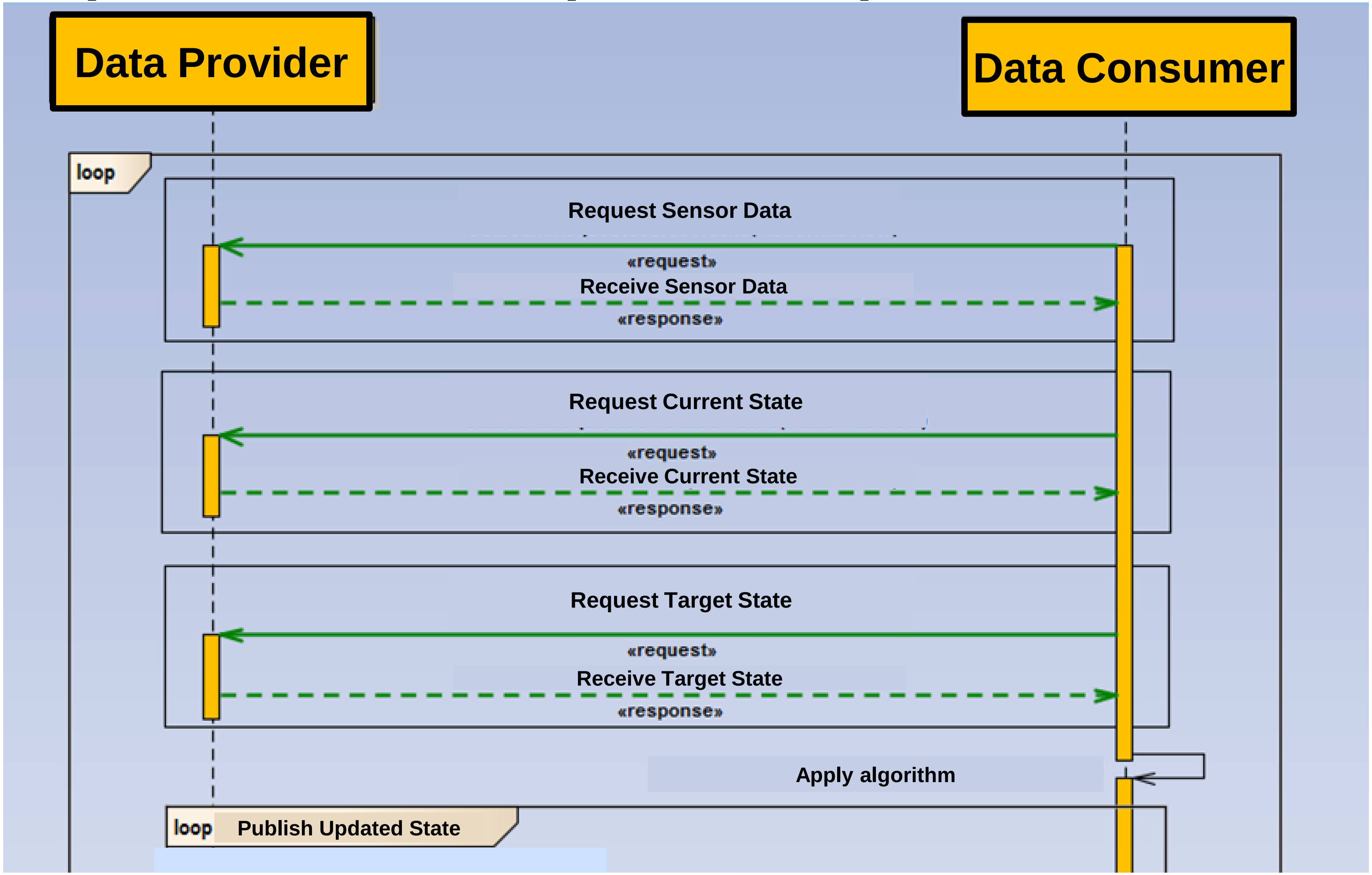
System implementation

This aircraft industry experience has led to the System Architecture Virtual Integration (SAVI) initiative

Why UML, SysML Are Not Sufficient

- System engineering
  - Focus on system architecture and operational environment
  - SysML developed to capture interactions with outside world, as a standardized UML profile
  - 4 pillars/diagrams: requirements, parameterics (added in SysML), structure, behavior
- Conceptual architecture
  - UML-based component model
  - Architecture views (DoDAF, IEEE 1471)
  - Platform Independent model (PIM)
- Embedded software system engineering
  - OMG Modeling and Analysis of Real Time Embedded systems (MARTE) as UML profile
    - Borrowed Meta model concepts from AADL
    - Focus on modeling implementations
  - xUML insufficient for PSM (Kennedy-Carter, NATO ALWI study)

Impact of Three Step Data Request Protocol



Operating as ARINC653 Partitioned System

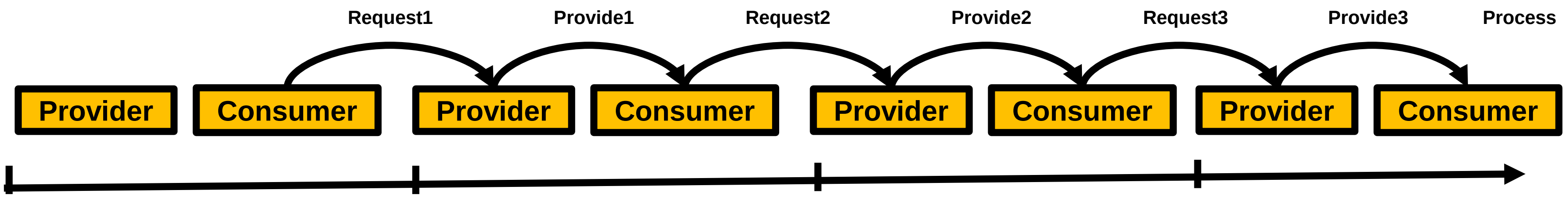
Data Consumer Requirement

- Process data in 1 second

Partitions

- Provide space and time boundary enforcement
- Execute periodically on a static timeline at 1 second rate

Data request protocols across partitions



How much time does consumer actually have to process the data?

Who pays for the communication overhead?

Model-based Engineering in Practice

Modeling is used in practice

- Modeling, analysis, and simulation in mechanical, control, computer hardware engineering

Current practice: modeling and software

  – Remember software through pictures
  – MDE and MDA with UML
  – Automatically generated documents

We need language for architecture modeling

- Strongly typed
- Well-defined execution and communication timing semantics
- Systematic approach to dealing with exceptional conditions
- Support for large-scale development

Outline

The Roots of AADL

1990-1998: MetaH and Control-H by Steve Vestal

- Strong typing, syntax borrowed from Ada
- Data and event ports, Operational modes
- Scheduling analysis and code generation

1994: Application to Missile Guidance System by Vestal and Lewis

- Three Week Port to Dual Processor Hardware

1997: MetaH Style for ACME by Peter Feiler and Jun Li

- CMU ECE Ph.D. on multi-dimensional analysis for Simplex architectures

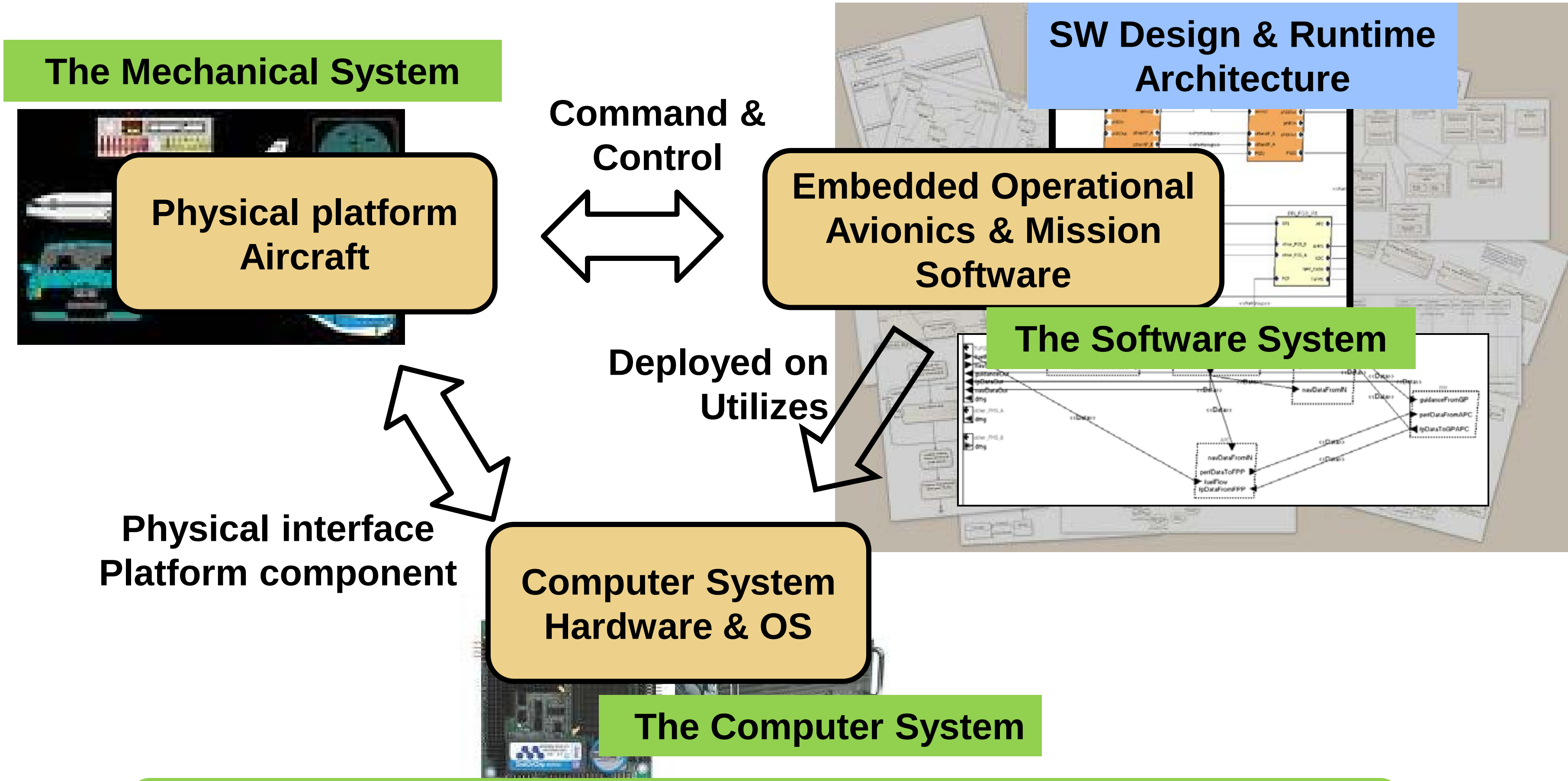
1998: Error Model added to MetaH by Steve Vestal

- Generation of fault trees and Markov models

1999: Requirements Document for AADL Standard

- Industry input: packages, messages
- The best of MetaH and ACME

SAE Architecture Analysis & Design Language (AADL) for Software-reliant Systems



AADL focuses on interaction between the three elements of a software-reliant mission and safety-critical systems.

The SAE AADL Standard Suite (AS-5506 series)

Core AADL language standard (V2.1-Sep 2012, V1-Nov 2004)

- Strongly typed language with well-defined semantics
- Textual and graphical notation
- Standardized XMI interchange format

Standardized AADL Extensions

Error Model language for safety, reliability, security analysis
ARINC653 extension for partitioned architectures
Behavior Specification Language for modes and interaction behavior
Data Modeling extension for interfacing with data models (UML, ASN.1, …)

AADL Annex Extensions in Progress

Requirements Definition and Assurance Annex
Synchronous System Specification Annex
Hybrid System Specification Annex
System Constraint Specification Annex
Network Specification Annex

System Level Fault Root Causes

Violation of data stream assumptions

- Stream miss rates, Mismatched data representation, Latency jitter & age

End-to-end latency analysis
Port connection consistency

Partitions as Isolation Regions

- Space, time, and bandwidth partitioning
- Isolation not guaranteed due to undocumented resource sharing
- fault containment, security levels, safety levels, distribution

Process and virtual processor to model partitioned architectures

Virtualization of time & resources

- Logical vs. physical redundancy
- Time stamping of data & asynchronous systems

Virtual processors & buses
Multiple time domains

Inconsistent System States & Interactions

- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

Operational and failure modes
Interaction behavior specification
Dynamic reconfiguration
Fault detection, isolation, recovery

Performance impedance mismatches

- Processor, memory & network resources
- Compositional & replacement performance mismatches
- Unmanaged computer system resources

Resource allocation & deployment configurations
Resource budget analysis & scheduling analysis

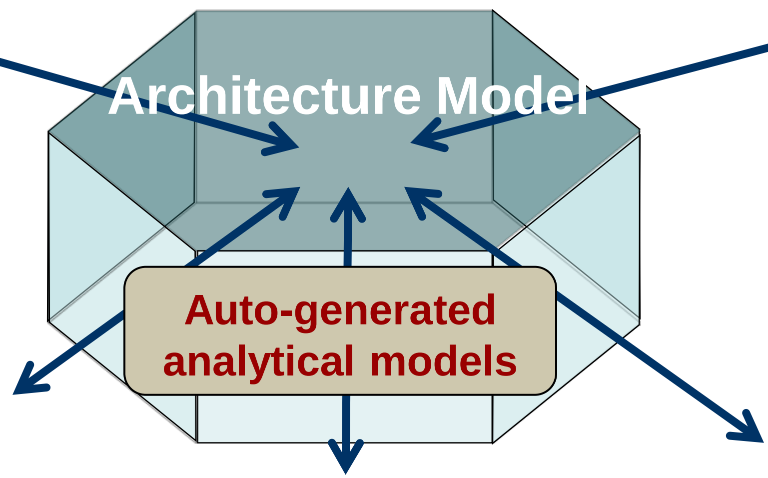
Codified in Virtual Upgrade Validation method

Architecture-Centric Quality Attribute Analysis

Single Annotated Architecture Model Addresses Impact Across Operational Quality Attributes

Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

Security
- Intrusion
- Integrity
- Confidentiality



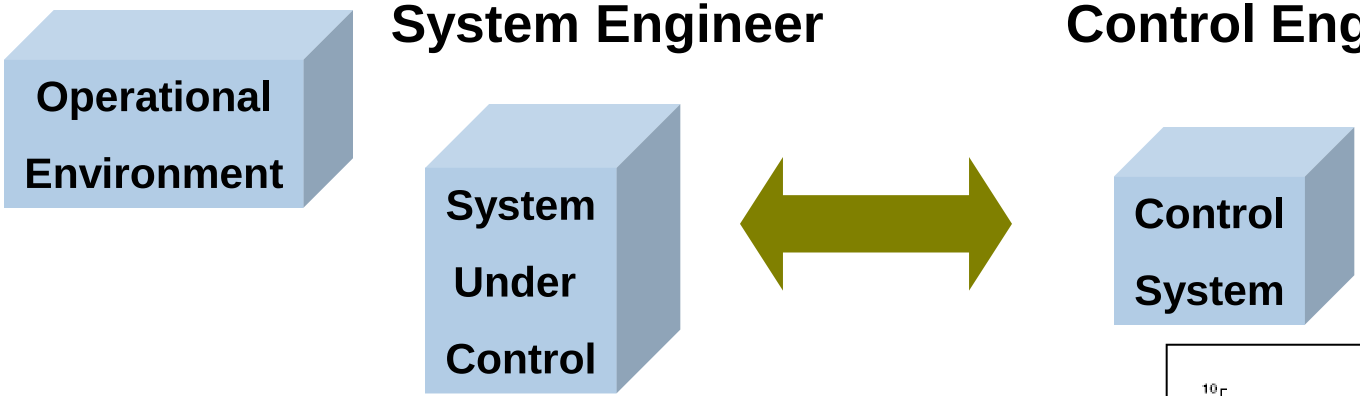
Data Quality
- Data precision/ accuracy
- Temporal correctness
- Confidence

Real-time Performance
- Execution time/ Deadline
- Deadlock/starvation
- Latency

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

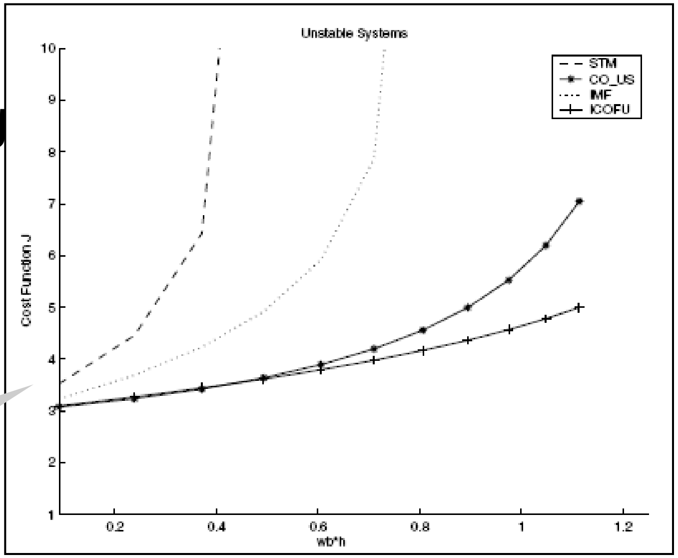
Multi-Fidelity End-to-end Latency in Control Systems

System Engineer **Control Engineer**



Common latency data from system engineering

- Processing latency
- Sampling latency
- Physical signal latency



Impact of Scheduler Choice on Controller Stability

A. Cervin, Lund U., CCACSD 2006

Software-Based Latency Contributors

Execution time variation: algorithm, use of cache

Processor speed

Resource contention

Preemption

Legacy & shared variable communicatior

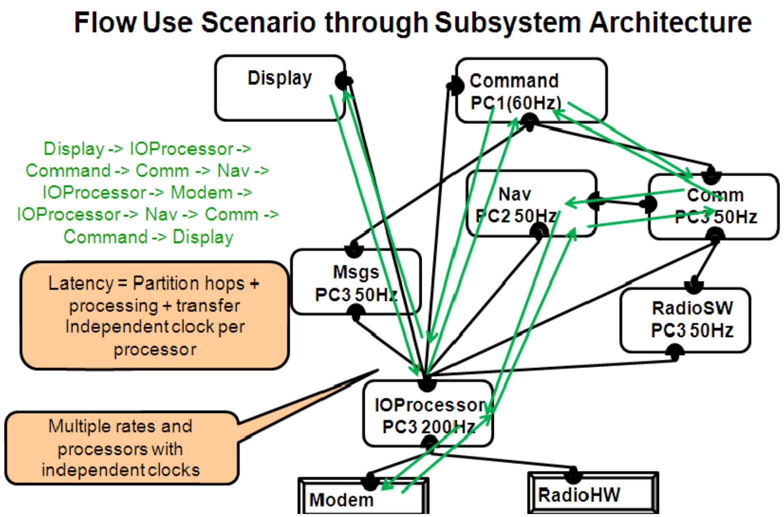
Rate group optimization

Protocol specific communication delay

Partitioned architecture

Migration of functionality

Fault tolerance strategy



Sampling of International Efforts Leveraging SAE AADL

Compositional Timing Framework
OSD 2014

P Project
Auto Code Gen
2011-2014

OPEES
Formal analysis
2011-2014

OpenGroup
Real-Time Forum
EU + US partners
2008-current

D-MILS
Design of Secure Systems
2013 - $4.9M

ESA COMPASS
System SW Co-engineering
2008-current

TOPCASED
Open Source Embedded Systems Tool Framework
28 partners €20+M 2005-2009

IST ARTIST2
Embedded Systems Center of Excellence
2007-2012

AVSI SAVI
Analysis-based System Validation
12 partners $20M 2008-current

ITEA SPICES
Model-Driven Embedded Systems Engineering
15 partners €16M 2006-2009

DARPA META
Complex System Engineering
2010-2012

Flex-eWare
Auto Code Generation
2007-2010

EC ASSERT
Proof-based Satellite Architectures
ESA + 30 partners
€15M 2004-200

ESA TASTE
System & SW Validation & Generation
2010-current

PARSEC
Safety/security f
2010-2013

DARPA HACMS
Security in CPS
RC formal methods
2013-2015

AADL Inspector
Ellidiss
2010-current

Integrated Clinical Environment
Device Certification
FDA KSU
2011-current

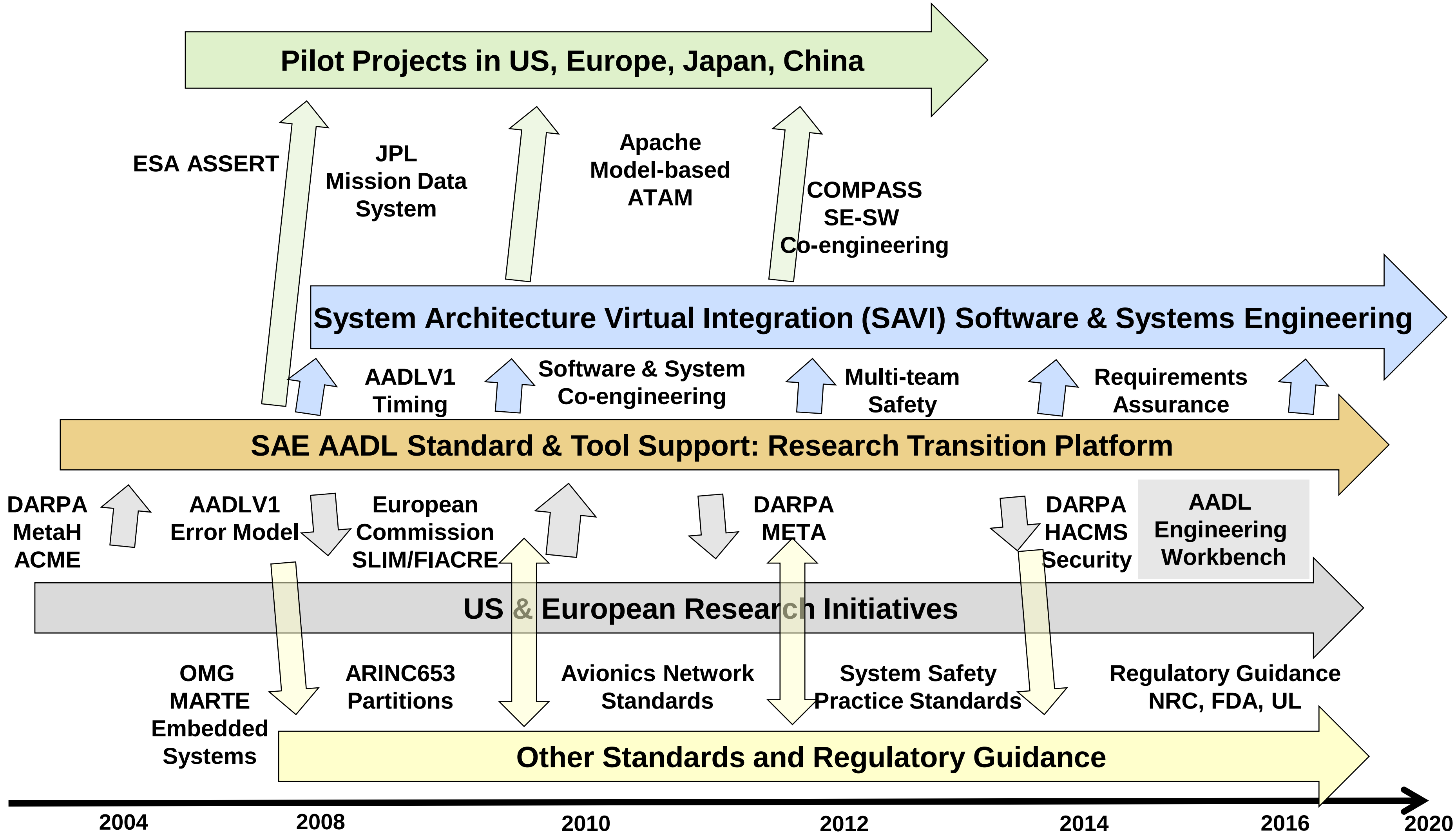
MASIW
Avionics Workbench
2011-current
$2M per year

PROARTIS
Partitioned RT systems
2010-2013 € 1.8M

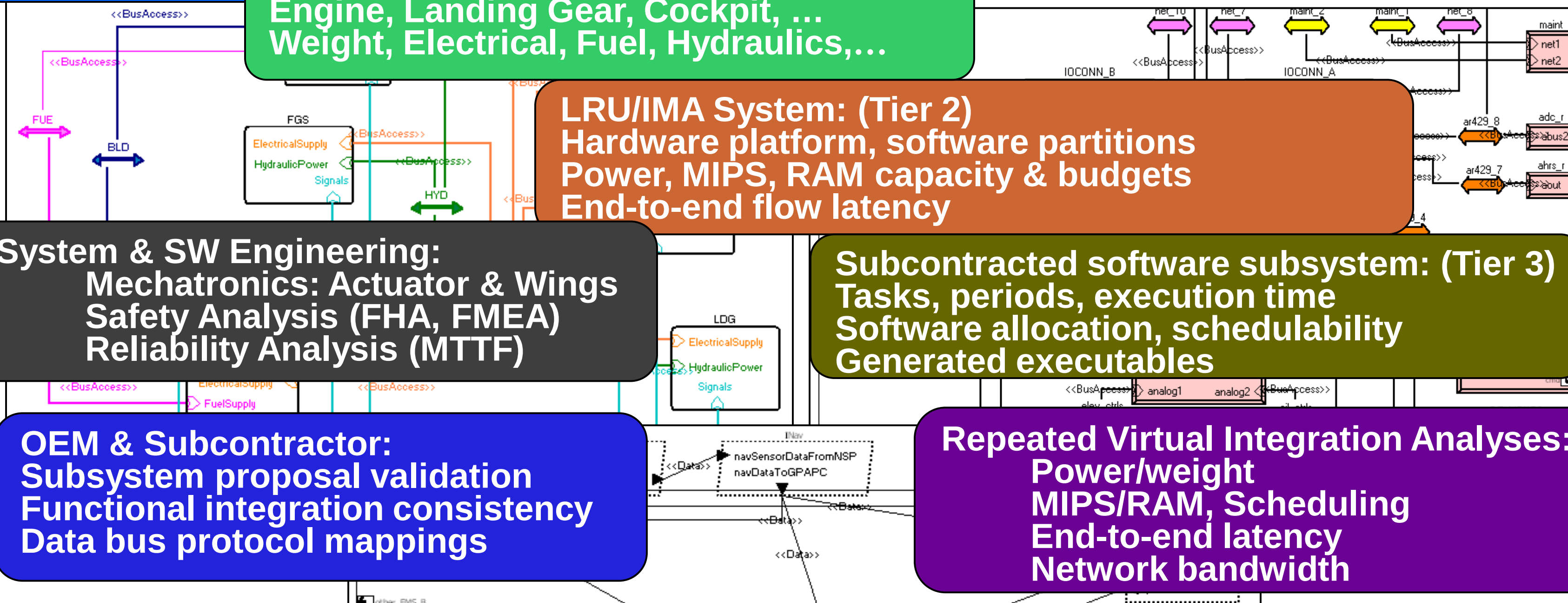
RAMSES
Auto Code Generation
2012-current

Architecture-centric Virtual System Integration

Evolution, Maturation and Transition



Early Discovery and Incremental V&V through System Architecture Virtual Integration (SAVI)



Aircraft: (Tier 0)

Aircraft system: (Tier 1)
Engine, Landing Gear, Cockpit, …
Weight, Electrical, Fuel, Hydraulics,…

LRU/IMA System: (Tier 2)
Hardware platform, software partitions
Power, MIPS, RAM capacity & budgets
End-to-end flow latency

System & SW Engineering:
- **Mechatronics: Actuator & Wings**
- **Safety Analysis (FHA, FMEA)**
- **Reliability Analysis (MTTF)**

Subcontracted software subsystem: (Tier 3)
Tasks, periods, execution time
Software allocation, schedulability
Generated executables

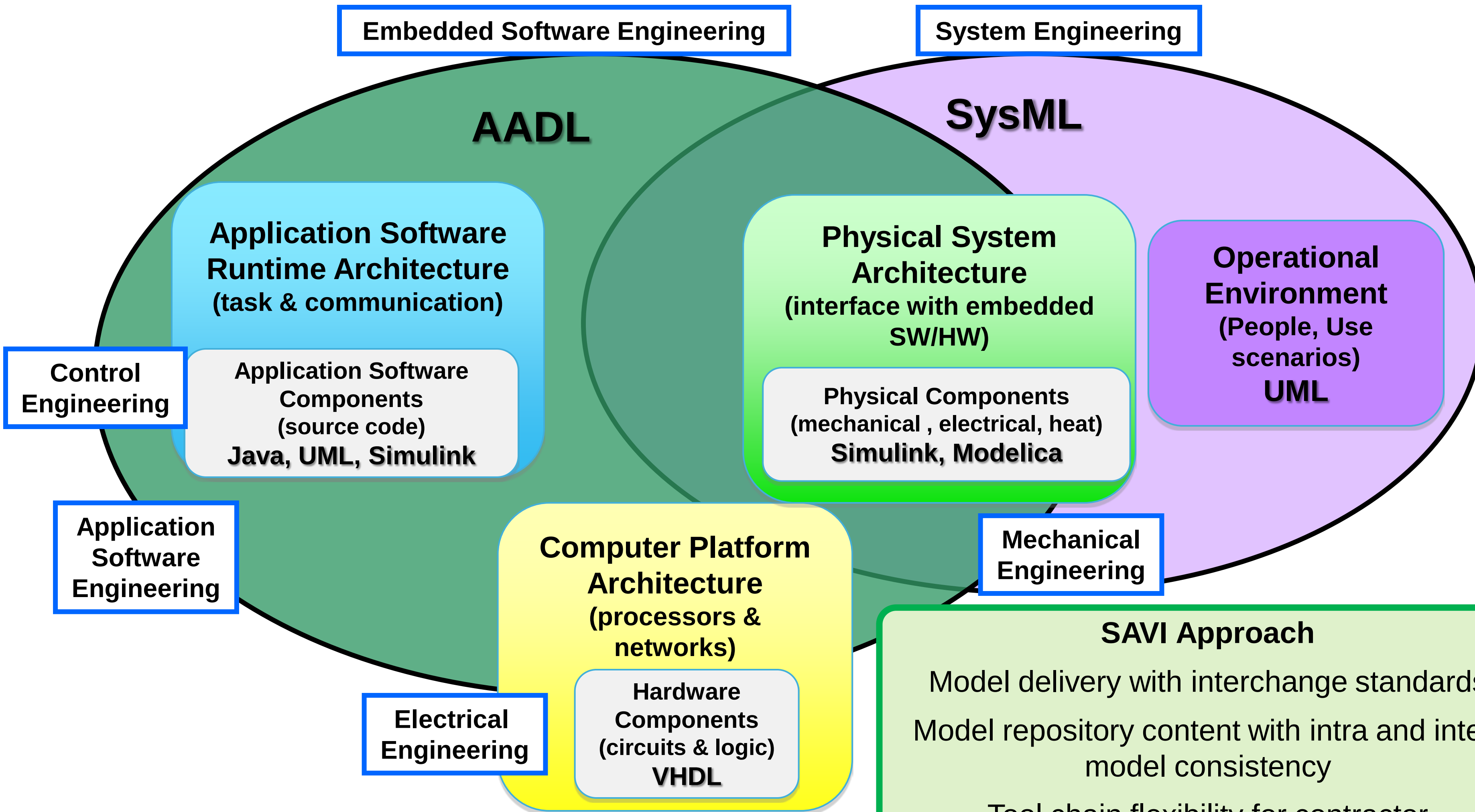
OEM & Subcontractor:
Subsystem proposal validation
Functional integration consistency
Data bus protocol mappings

Repeated Virtual Integration Analyses:
- **Power/weight**
- **MIPS/RAM, Scheduling**
- **End-to-end latency**
- **Network bandwidth**

Proof of Concept Demonstration and Transition by Aerospace industry initiative
- **Architecture-centric model-based software and system engineering**
- **Architecture-centric model-based acquisition and development process**
- **Multi notation, multi team model repository & standardized model interchange**

- Multi-tier system & software architecture (in AADL)
- Incremental end-to-end validation of system properties

Multi-Notation Approach to Architecture-centric Virtual System and Software Integration



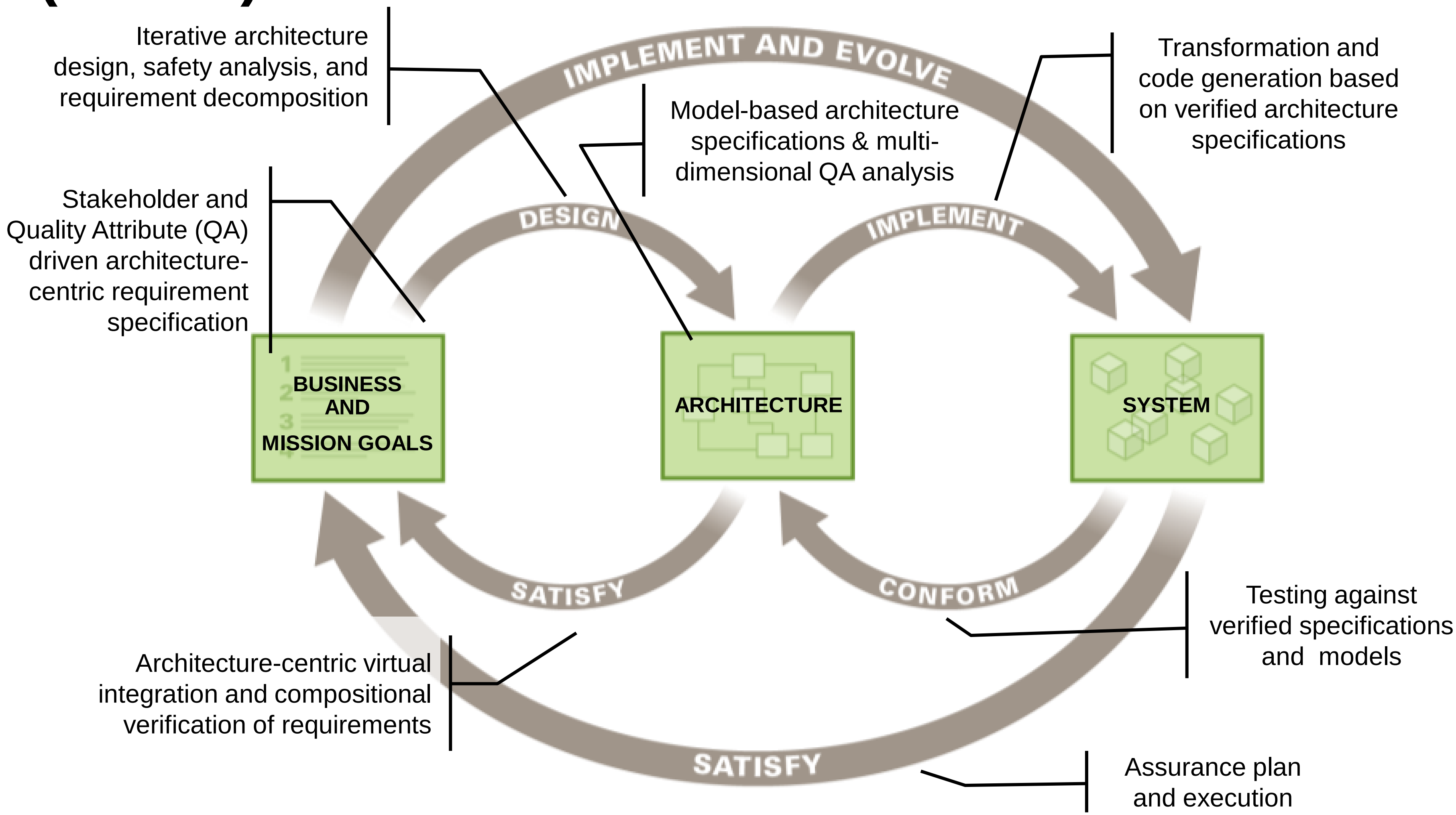
SAVI Approach

Model delivery with interchange standards

Model repository content with intra and inter-model consistency

Tool chain flexibility for contractor

Architecture-centric Virtual Integration Practice (ACVIP)



Outline

Challenges in Safety-critical Software-intensive systems

An Architecture-centric Virtual Integration Strategy with SAE AADL

Improving the Quality of Requirements

Architecture Fault Modeling and Safety

Incremental Life-cycle Assurance of Systems

Summary and Conclusion

Certification & Recertification Challenges

Certification: assure the quality of the delivered system

- Sufficient evidence that a system implementation meets system requirements
- Quality of requirements and quality of evidence determines quality of system

Certification related rework cost

- Currently 50% of total system cost and growing

Recertification Challenge

- Desired cost of recertification in proportion to change

Improve quality of requirements and evidence

Perform verification compositionally throughout the life cycle

Current Industry Practice in DO-178B Compliant Requirements Capture

Industry Survey in 2009 FAA Requirements Engineering Study

Notation

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requireme	Low-Level Software Requireme	Hardware Requirements
English Text or Shall Statements	39	27	36	32	29
Tables and Diagrams	31	30	30	19	18
UML Use Cases	1		2	4	
UML Sequence Diagrams			3	6	
UML State Diagrams			1	7	
Executable Models (e.g. Simulink, SCADE Suite, etc.)	7	1	8	8	1
Data Flow Diagrams (e.g. Yourdon)	4		6	9	
Other (Specify)XML		1			
Operational models or prototypes	1	1			1
UML			1	1	

Need analyzable & executable specifications

Tool

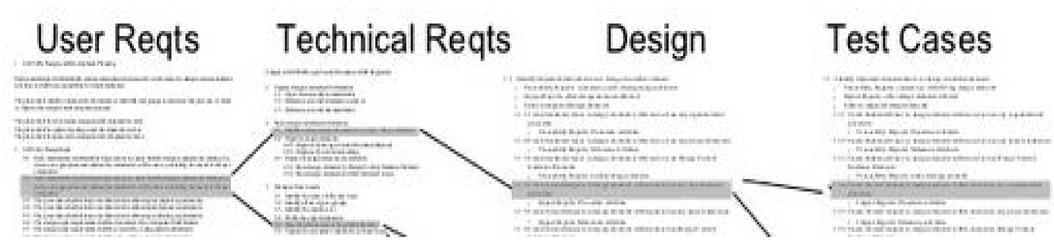
Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Database (e.g., Microsoft Access)	3	4	3	3	
DOORS	23	13	22	18	12
Rational ROSE®			1	3	
RDD-100®					
Requisite Pro®	5	3	5	4	4
Rhapsody	1				
SCADE Suite	2		3	1	
Simulink	5	1	5	3	1
Slate	1		1	1	
Spreadsheet (e.g., Microsoft Excel)	5	4	5	4	3
Statemate					
Word Processor (e.g., Microsoft Word)	19	20	18	17	16
VAPS™		1	3	3	
Designer's Workbench™			1	1	
Proprietary Database, SCADE like pic tool		1	1		
Interleaf	1	1	1	1	1
BEACON	1	1	1	1	
CaliberRM	1	1	1	1	1
XM:		1			
Wiring diagram		1			1

Requirement Quality Challenge

Requirements error	%
Incomplete	21%
Missing	33%
Incorrect	24%
Ambiguous	6%
Inconsistent	5%

There is more to requirements quality than "shall"s and stakeholder traceability

IEEE 830-1998 Recommended Practice for SW Requirements Specification



Browsable links/Coverage metrics

IEEE Std 830-1998 characteristics of a good requirements specification:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

System to SW requirements gap [Boehm 2006]

How do we verify low level SW requirements against system requirements?

When StartUpComplete is TRUE in both FADECs and SlowStartupComplete is FALSE, the FADECStartupSW shall set SlowStartupInComplete to TRUE

Mixture of Requirements & Architecture Design Constraints

Requirements for a Patient Therapy System

The patient shall never be infused with a single air bubble more than 5ml volume.

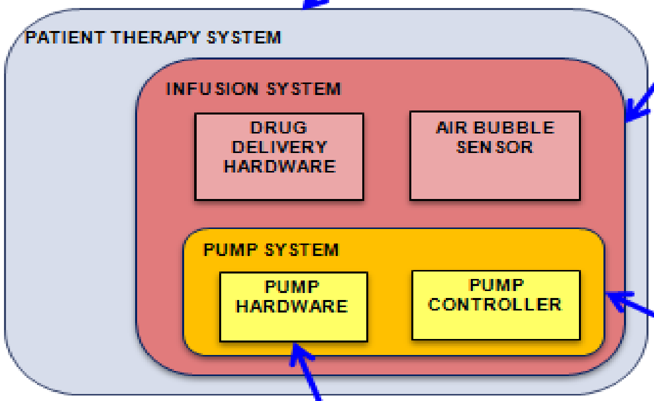
When a single air bubble more than 5ml volume is detected, the **system** shall stop infusion within 0.2 seconds.

When piston stop is received, the **system** shall stop piston movement within 0.01 seconds.

The **system** shall always stop the piston at the bottom or top of the chamber.

Requirements and Design Information

1. The patient shall never be infused with a single air bubble more than 5ml volume.
2. When a single air bubble more than 5ml volume is detected, the **system** shall stop infusion within 0.2 seconds.



3. When piston stop is received, the **system** shall stop piston movement within 0.01 seconds.
4. The **system** shall always stop the piston at the bottom or top of the chamber.

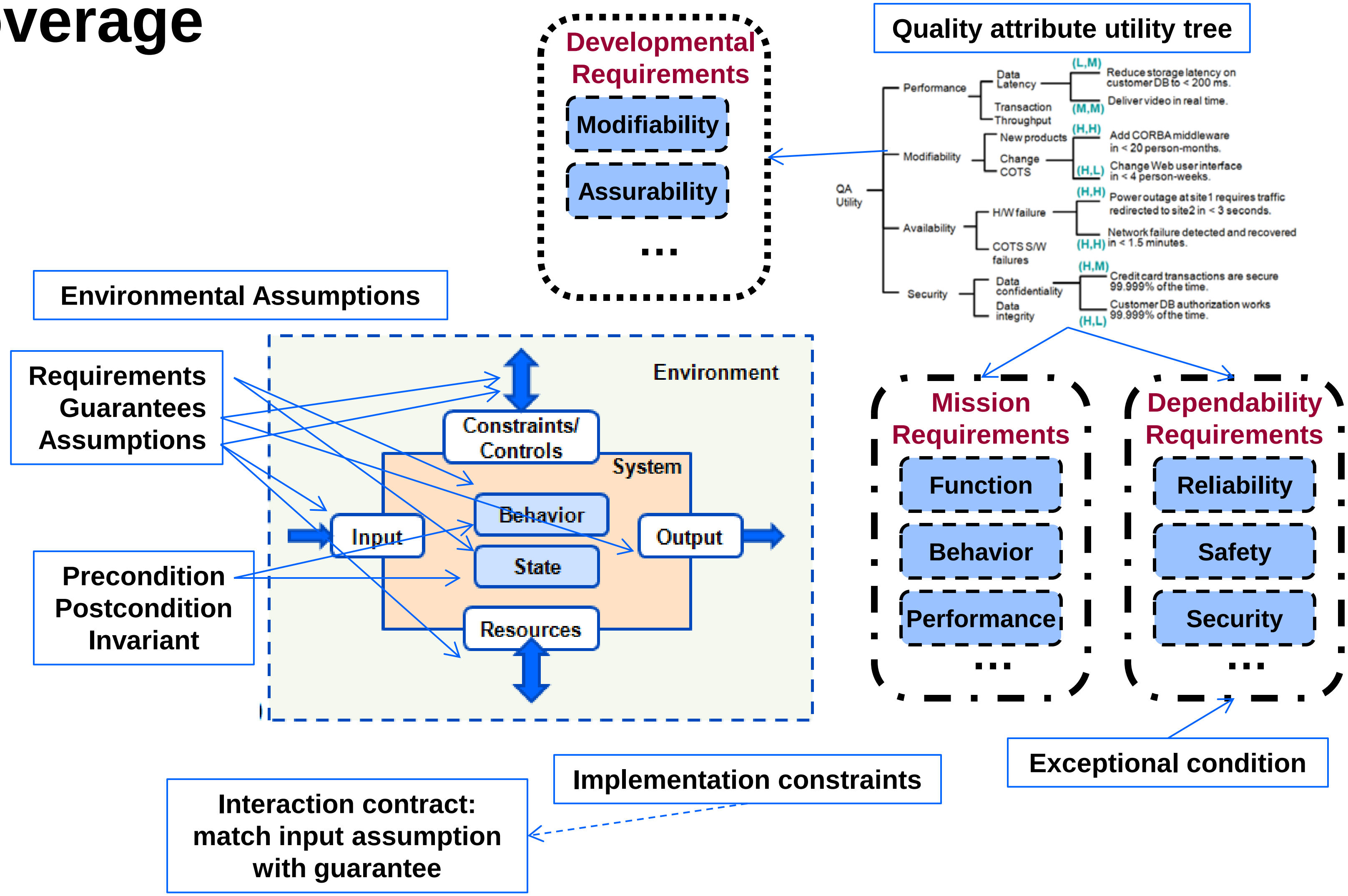
Typical requirement documents span multiple levels of a system architecture

We have made architecture design decisions.

We have effectively *specified a <u>partial architecture</u>*

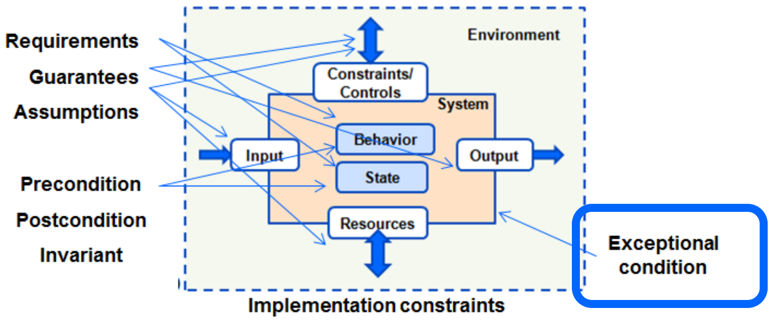
Adapted from M. Whalen presentation

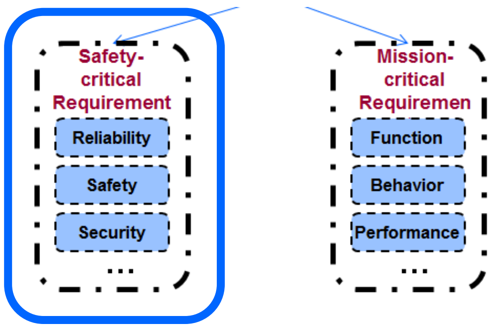
System Specification and Requirements Coverage



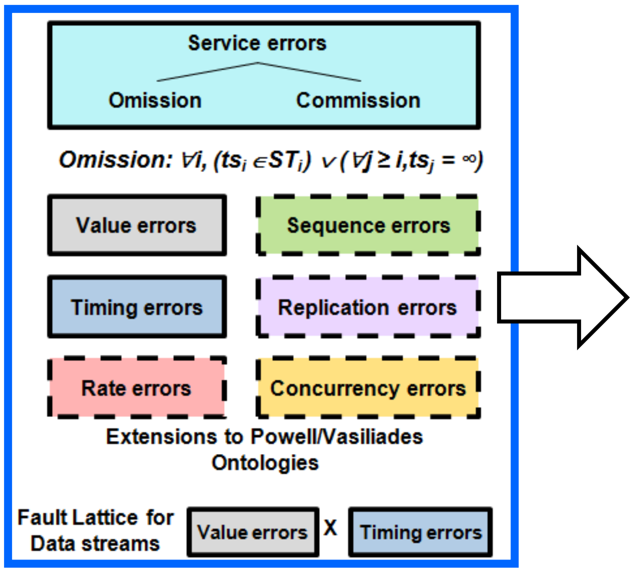
Architecture-led Requirement & Hazard Specification

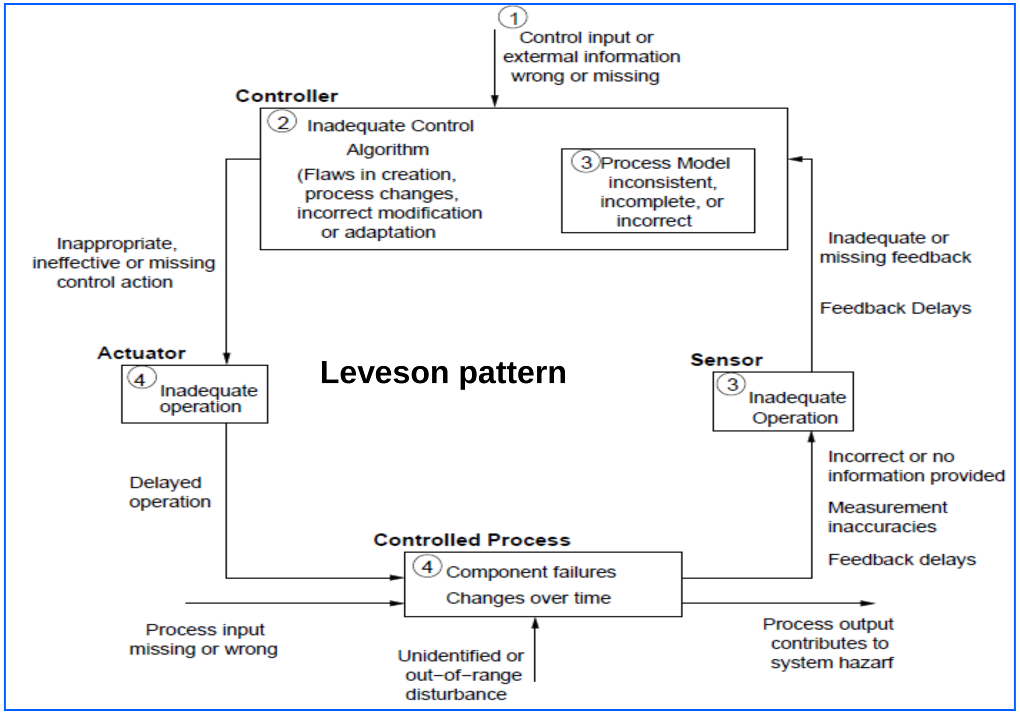
System Specification Coverage





Error Propagation Ontology



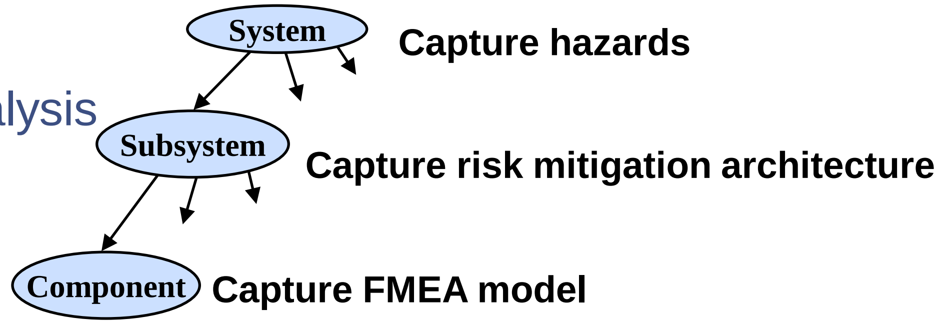


Outline

AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.

- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes



Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.

Annotated architecture model permits checking for consistency and completeness between these various declarations.

Related analyses are also useful for other purposes, e.g.

- maintainability
- availability
- Integrity
- Security

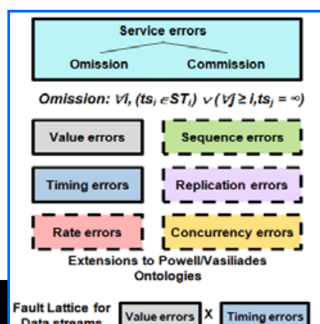
SAE ARP 4761 *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*
Demonstrated in SAVI Wheel Braking System Example

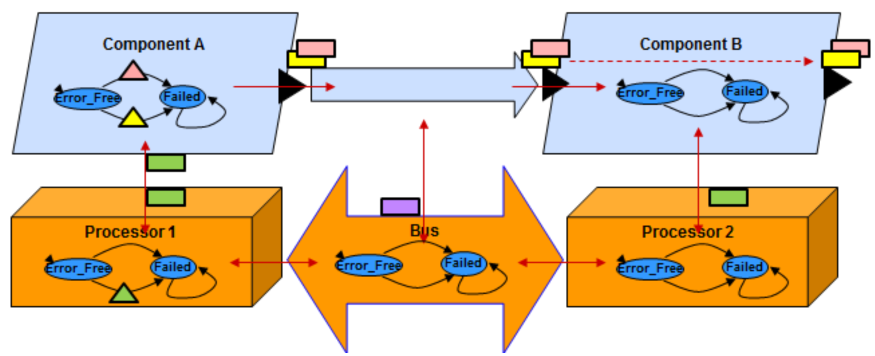
Error Model Annex can be adapted to other ADLs

Error Model V2: Abstraction and Refinement

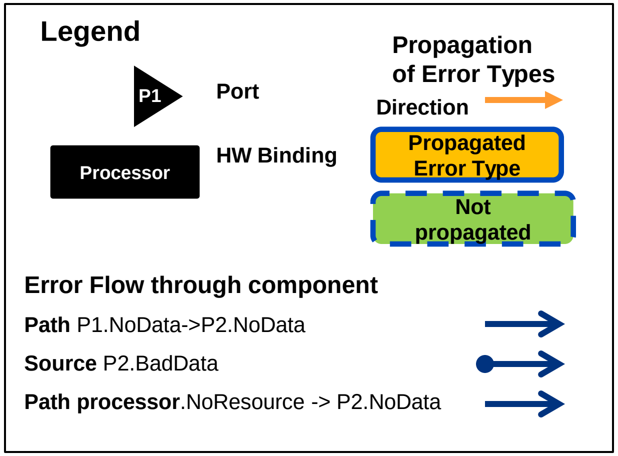
Four levels of abstraction:

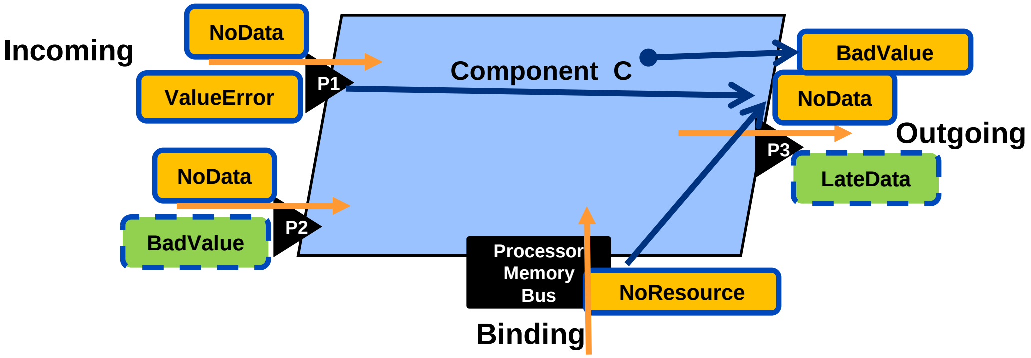
- Focus on fault interaction with other components
  - Probabilistic error sources, sinks, paths and transformations
  - Fault propagation and Transformation Calculus (FPTC) from York U.
- Focus on fault behavior of components
  - Probabilistic typed error events, error states, propagations
  - Voting logic, error detection, recovery, repair
- Focus on fault behavior in terms of subcomponent fault behaviors
  - Composite error behavior state logic maps states of parts into (abstracted) states of composite
- Types of malfunctions and propagations
  - Common fault ontology





Error Propagation Contracts





"Not" on propagated indicates that this error type is intended to be contained.

This allows us to determine whether propagation specification is complete.

Incoming/Assumed

- **Error Propagation**
  Propagated errors
- **Error Containment:**
  Errors not propagated

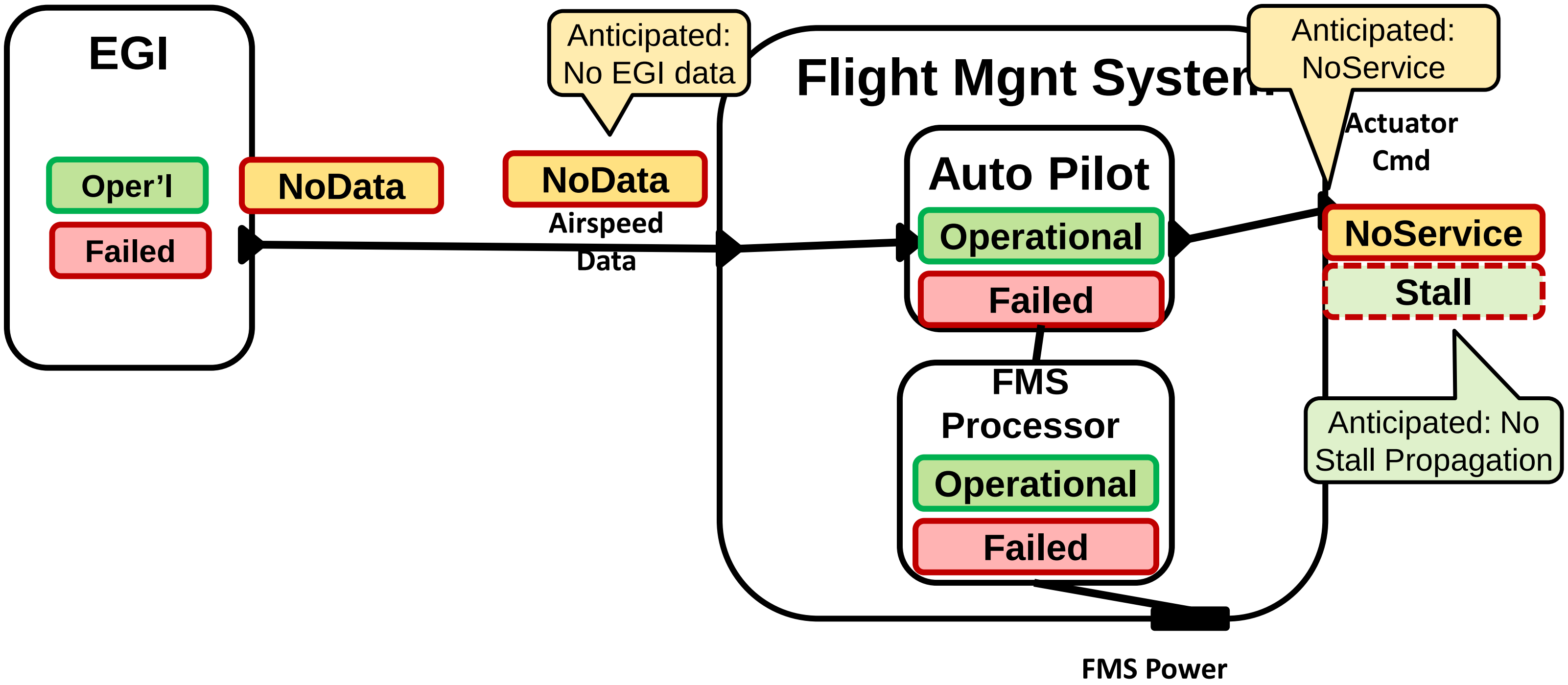
Outgoing/Contract

- **Error Propagation**
- **Error Containment**

Bound resources

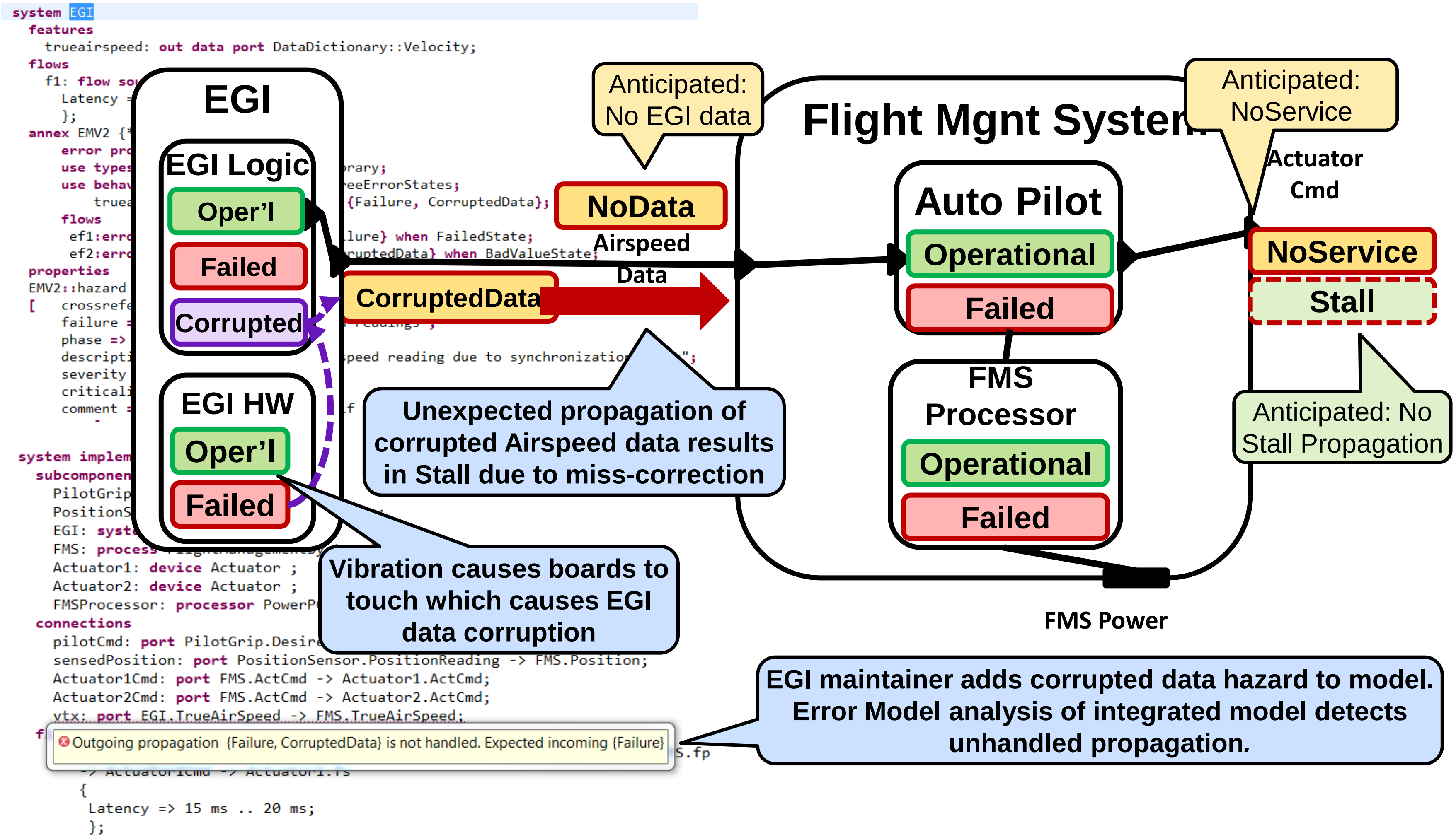
- **Error Propagation**
- **Error Containment**
- **Propagation to resource**

Original Preliminary System Safety Analysis (PSSA)



System engineering activity with focus on failing components.

Discovery of Unexpected PSSA Hazard through Repeated Virtual Integration



Recent Automated FMEA Experience

Failure Modes and Effects Analyses are rigorous and comprehensive reliability and safety design evaluations

- Required by industry standards and Government policies
- When performed manually are usually done once due to cost and schedule
- If automated allows for
  - multiple iterations from conceptual to detailed design
  - Tradeoff studies and evaluation of alternatives
  - Early identification of potential problems

ID	Item	Initial State	Initial Failure Mode	1st Level Effect	Transition	2nd Level Effect	Transition	3rd Level Effect	Severity	M
1	Sat_Bus	Working	Failure	Failed		Failed	Recovery	Working		Workir
1	Sat_Payload	Working		Working	Bus failure causes payload transition	Standby		Standby	Bus Recovery Causes Payload Transition	Workir
2	Sat_Bus	Working		Working		Working	5			
2	Sat_Payload	Working	Failure	Failed	Recovery	Working	5			

Largest analysis of satellite to date consists of 26,000 failure modes

- Includes detailed model of satellite bus
- 20 states perform failure mode
- Longest failure mode sequences have 25 transitions (i.e., 25 effects)

Myron Hecht, Aerospace Corp.
Safety Analysis for JPL, member of DO-178C committee

Support of SAE ARP4761 System Safety Assessment Practice

AADL & EMV2

FHA

Spreadsheet

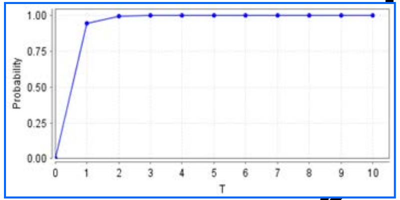
Uses error sources

Component	Error	Hazard Description	Crossrefer	Functional Failure	Operational P
StabilatorPositionSe	"ServiceOmission o	"No stabilator position readings due to s	"1.1.3"	"Loss of sensor readings"	"all"
StabAct1	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabAct2	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"

Markov Chain

PRISM

Uses error flows & behavior



FMEA

Spreadsheet

Uses error flows & propagations

Three CPU FMEA

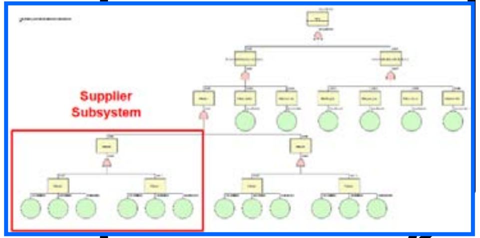
Date: 12/1/2010 Author:

Item	Initial State	Initial Failure	1st Level Effect	Transition	2nd Level Effect
CPU_1.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_L	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_2.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_R	ErrorFree		ErrorFree		ErrorFree
PR_FGS_L1	ErrorFree		ErrorFree		ErrorFree
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree
CPU_1.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_L	ErrorFree		ErrorFree		ErrorFree
CPU_2.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_R	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
PR_FGS_L1	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree

FTA

CAFTA, OpenFTA

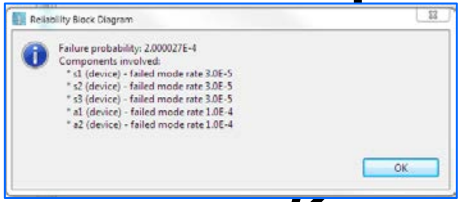
Uses composite error behavior



RBD/DD

OSATE plugin

Uses composite error behavior



The Symptom: Missed Stepper Motor Steps

Stepper motor (SM) controls a valve

- Commanded to achieve a specified valve position
  - Fixed position range mapped into units of SM steps
- New target positions can arrive at any time
  - SM immediately responds to the new desired position

Safety hazard due to software design

- Execution time variation results in missed steps
- Leads to misaligned stepper motor position and control system states
- Sensor feedback not granular enough to detect individual step misses

Software modeled and verified in SCADE

Full reliance on SCADE of SM & all functionality

Problems with missing steps not detected

Software tests did not discover the issue

Time sensitive systems are hard to test for.

Two Customer Proposed Solutions

Sending of data at 12ms offset from dispatch

Buffering of command by SM interface

No analytical confidence that the problem will be addressed

Other Challenge Problems

Aircraft wheel braking system

Engine control power up

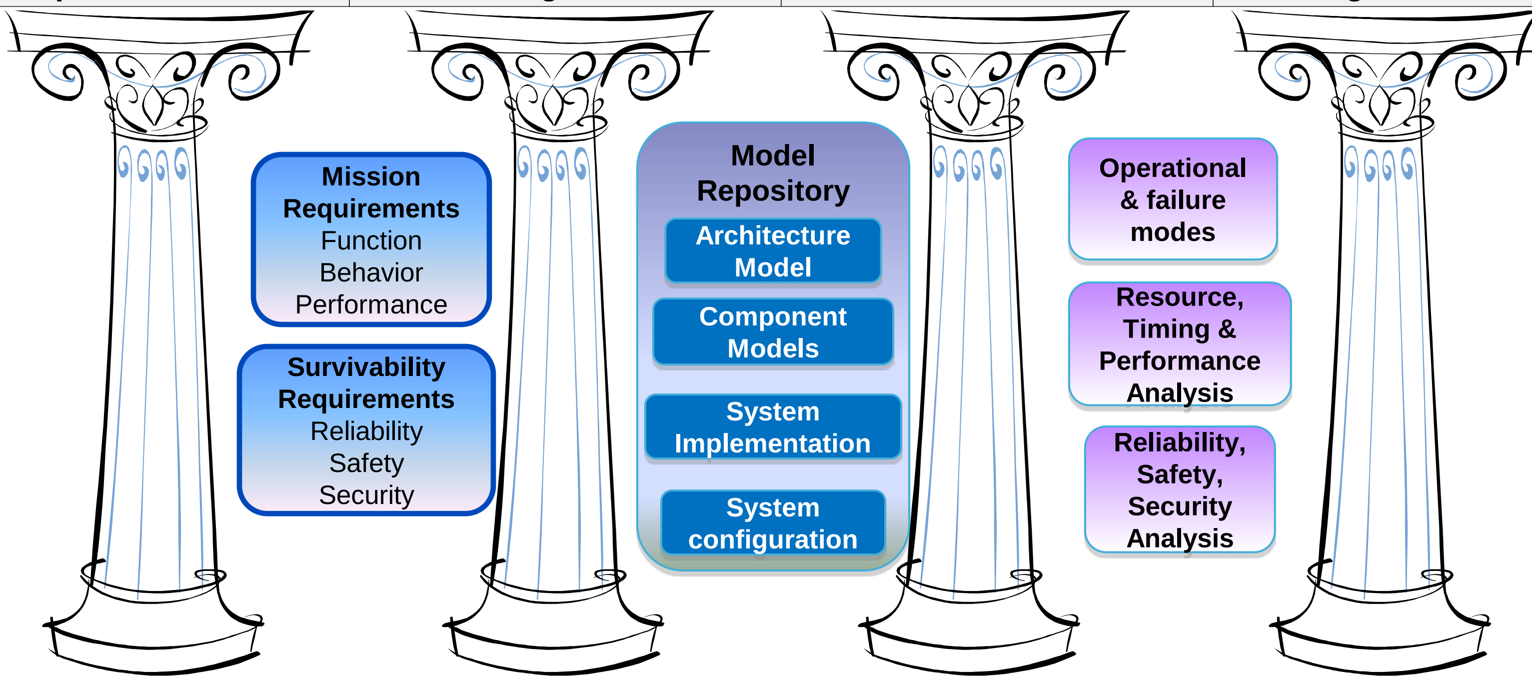
Situational Awareness & health monitoring

Outline

Reliability & Qualification Improvement Strategy

2010 SEI Study for AMRDEC
Aviation Engineering Directorate

Architecture-led Requirement Specification	Architecture-centric Virtual System Integration	Static Analysis & Compositional Verification	Incremental Assurance Plans & Cases throughout Life Cycle



Four pillars for Improving Quality of Critical Software-reliant Systems

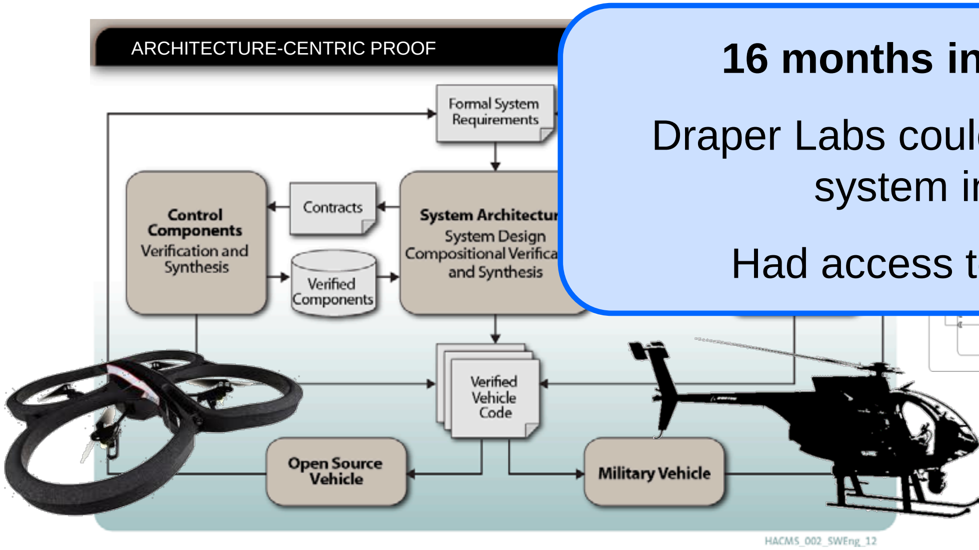
Contract-based Compositional Verification

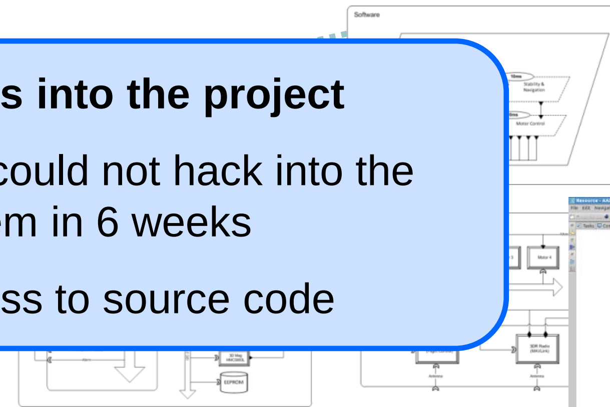
Secure Mathematically-Assured Composition of Control Models

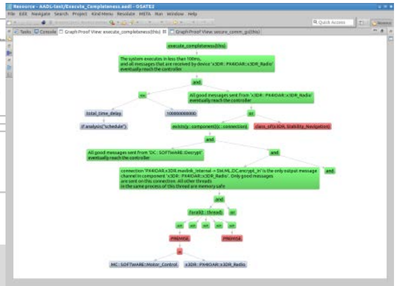
Key Problem
Many vulnerabilities occur at component interfaces. How can we use formal methods to detect these vulnerabilities and build provably secure systems?

TA4 – Research Integration and Formal Methods Workbench
Rockwell Collins and University of Minnesota

SMACCM







16 months into the project

Draper Labs could not hack into the system in 6 weeks

Had access to source code

Technical Approach

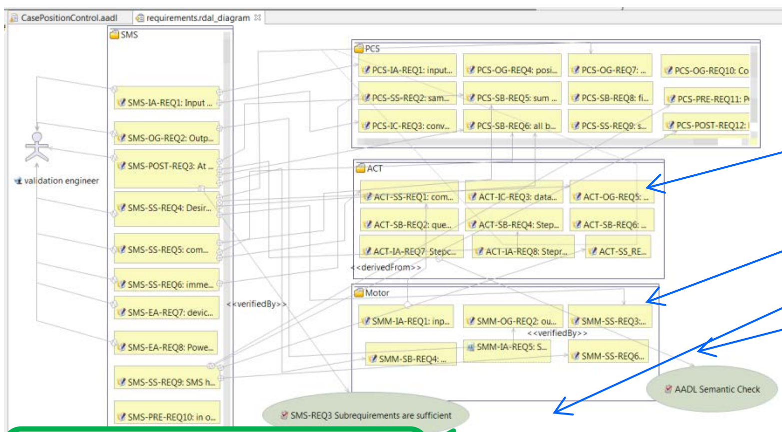
- Develop a complete, formal architecture model for UAVs that provides robustness against cyber attack
- Develop compositional verification tools driven from the architecture model for combining formal evidence from multiple sources, components, and subsystems
- Develop synthesis tools to generate flight software for UAVs directly from the architecture model, verified components, and verified operation system

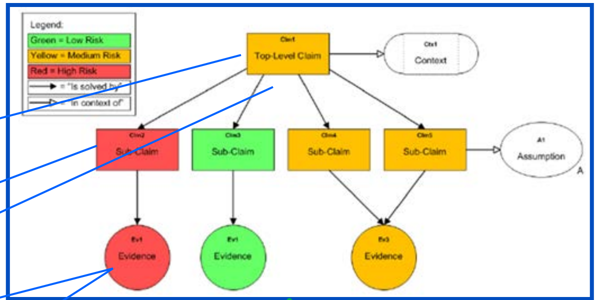
Accomplishments

- Created AADL model of vehicle hardware & software architecture
- Identified system-level requirements to be verified based on input from Red Team evaluations
- Developed Resolute analysis tool for capturing and evaluating assurance case arguments linked to AADL model
- Developed example assurance cases for two security requirements
- Developed synthesis tool for auto-generation of configuration data and glue code for OS and platform hardware

Open source tools available at github.com/smaccm

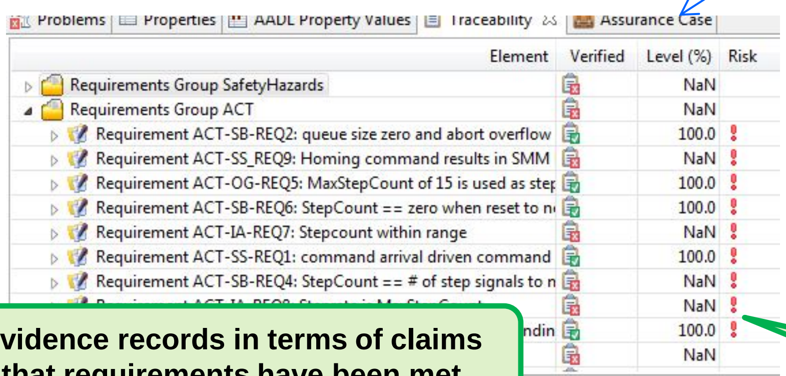
Integrated Approach to Requirement V&V through Assurance Automation



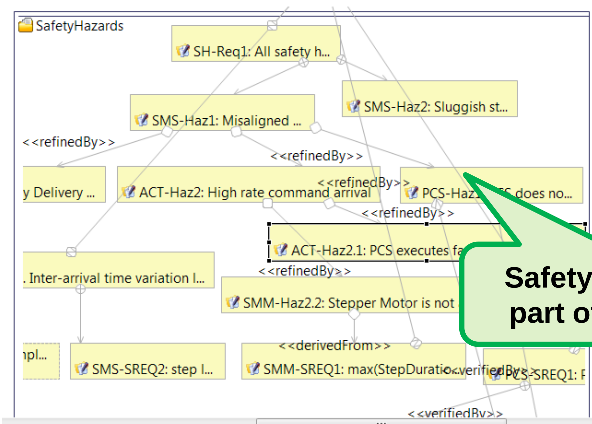


Generated assurance cases

Requirement coverage
Assumption evidence



Element	Verified	Level (%)	Risk
Requirements Group SafetyHazards		NaN	
Requirements Group ACT		NaN	
Requirement ACT-SB-REQ2: queue size zero and abort overflow		100.0	!
Requirement ACT-SS_REQ9: Homing command results in SMM		NaN	!
Requirement ACT-OG-REQ5: MaxStepCount of 15 is used as step		100.0	!
Requirement ACT-SB-REQ6: StepCount == zero when reset to n		100.0	!
Requirement ACT-IA-REQ7: Stepcount within range		NaN	!
Requirement ACT-SS-REQ1: command arrival driven command		100.0	!
Requirement ACT-SB-REQ4: StepCount == # of step signals to n		NaN	!
		NaN	!
ndin		100.0	!
		NaN	



Safety hazards are part of the picture

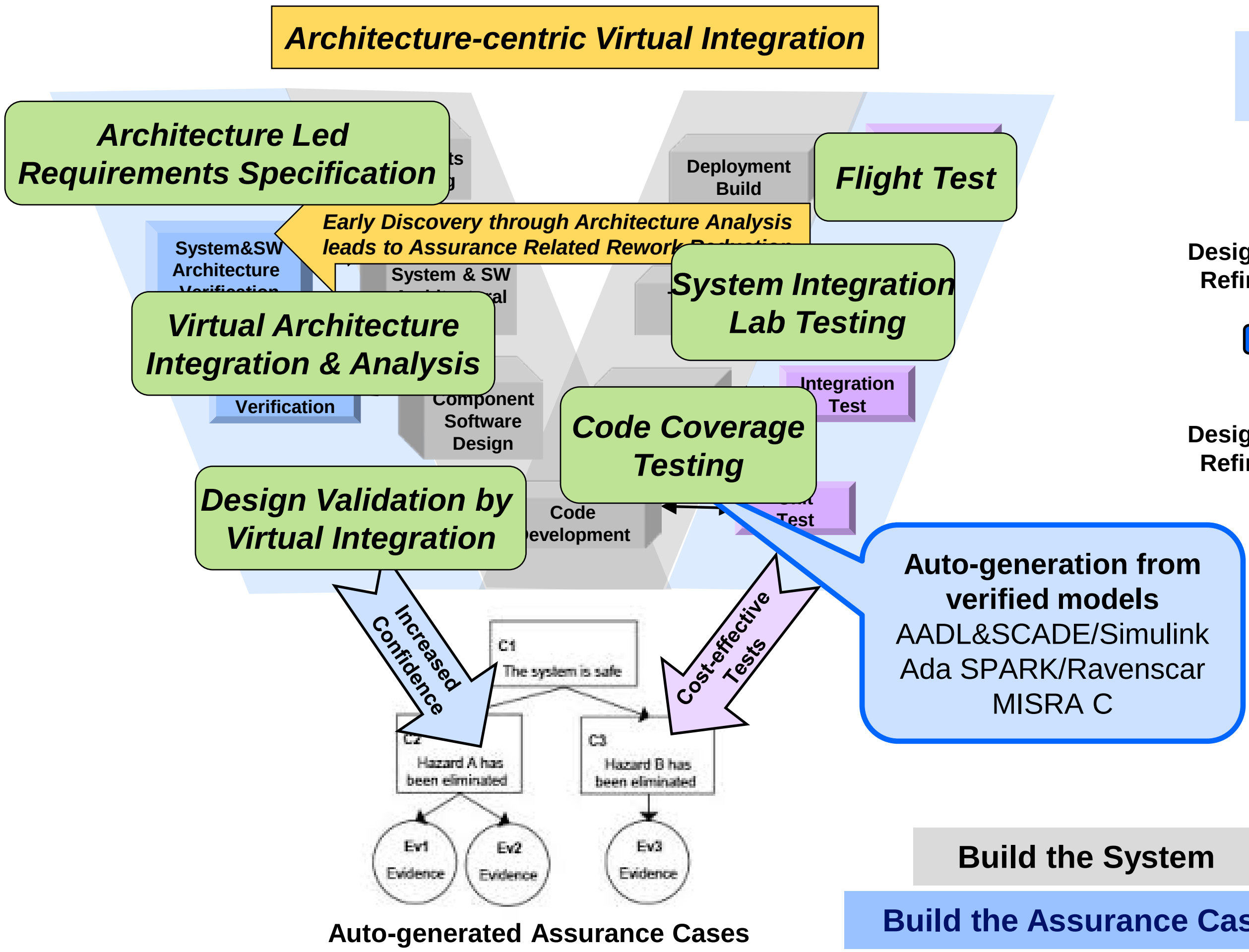
Evidence records in terms of claims that requirements have been met

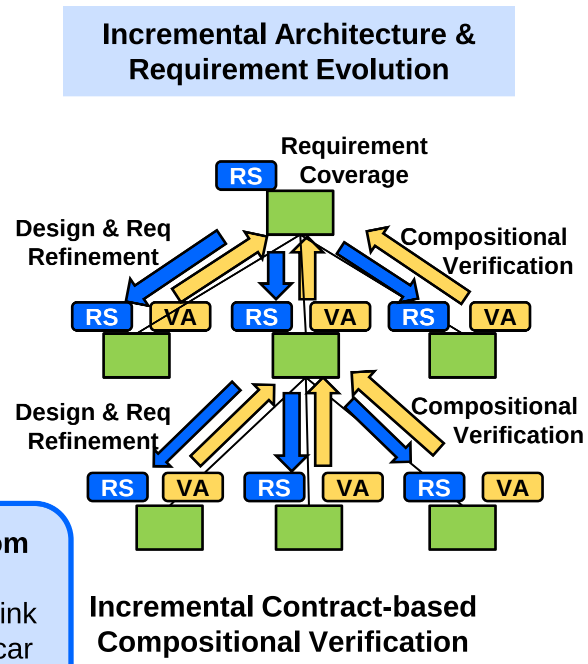
Linkage to automated test harnesses

Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Incremental Evolution and Execution of Assurance Plans





Build the System

Build the Assurance Case

Outline

Challenges in Safety-critical Software-intensive systems

An Architecture-centric Virtual Integration Strategy with SAE AADL

Improving the Quality of Requirements

Architecture Fault Modeling and Safety

Incremental Life-cycle Assurance of Systems

Summary and Conclusion

Benefits of Architecture-centric Engineering

Reduce risks

- Analyze system early and throughout life cycle
- Understand system wide impact
- Validate assumptions across system

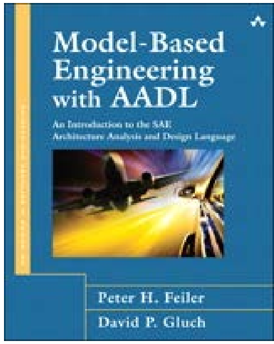
Increase confidence

- Validate models to complement integration testing
- Validate model assumptions in operational system
- Evolve system models in increasing fidelity

Reduce cost

- Fewer system integration problems
- Fewer validation steps through use of validated generators

References



AADL Website www.aadl.info and AADL Wiki www.aadl.info/wiki

Blog entries and podcasts on AADL at www.sei.cmu.edu

AADL Book in SEI Series of Addison-Wesley
http://www.informit.com/store/product.aspx?isbn=0321888944

On AADL and Model-based Engineering

http://www.sei.cmu.edu/library/assets/ResearchandTechnology_AADLandMBE.pdf

On an architecture-centric virtual integration practice and SAVI

http://www.sei.cmu.edu/architecture/research/model-based-engineering/virtual_system_integration.cfm

On an a four pillar improvement strategy for software system verification and qualification

http://blog.sei.cmu.edu/post.cfm/improving-safety-critical-systems-with-a-reliability-validation-improvement-framework

Webinars on system verification https://www.csiac.org/event/architecture-centric-virtual-integration-strategy-safety-critical-system-verification and on architecture trade studies with AADL https://www.webcaster4.com/Webcast/Page/139/5357

15 Years of the SAE AS-2C AADL Committee

10 Years since the first publication of the SAE AADL standard

And many more ☺

Contact Information

Peter H. Feiler
Principal Researcher
RTSS
Telephone: +1 412-268-7790
Email: phf@sei.cmu.edu

Web
Wiki.sei.cmu.edu/aadl
www.aadl.info

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257